ODSA Transaction and Link Layer Specification for BoW Interfaces

Revision A

Version 0.8.1

16 December 2022

Table of Contents

License

Introduction

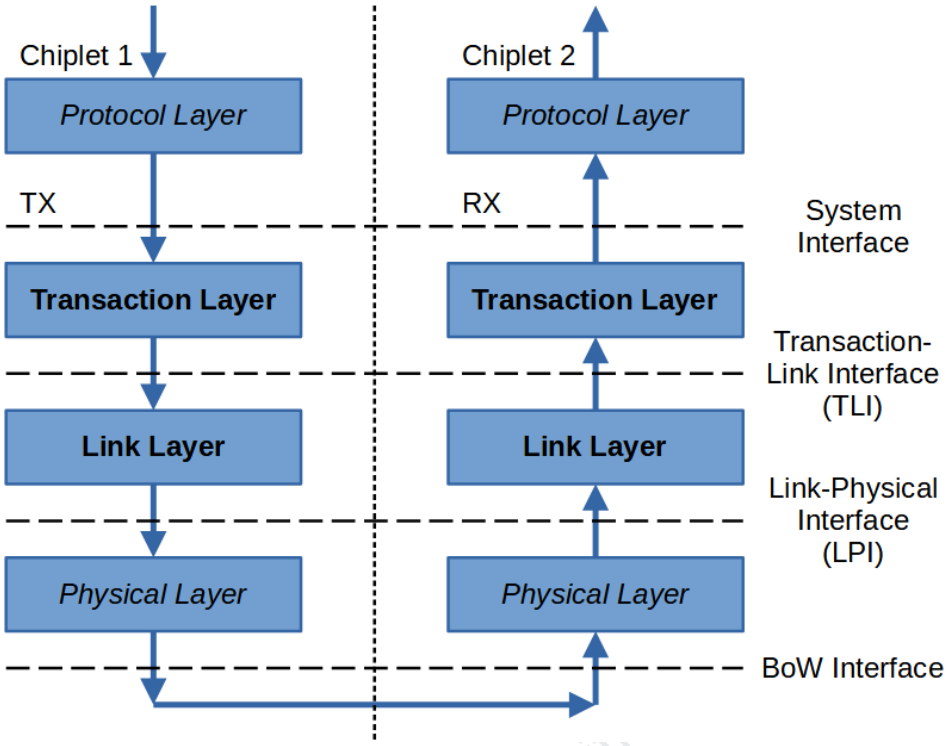
This specification describes Revision A (or Rev A) of the transaction and link layers for ODSA die-to-die (D2D) communication. The transaction and link layers form two parts of a layered architecture using the ODSA bunch-of-wires (BoW) electrical interface and consisting of the following:

- A *protocol layer* that defines communication between SoC IPs using industry-standard or proprietary protocols
- A *transaction layer* that translates between protocol transfers or protocol packets defined by a bus protocol and individual transaction streams and that manages the flow control of those individual streams
- A *link layer* that converts between the individual transaction streams and a single bitstream transmitted between chiplets
- A *physical layer* that performs the physical transmission of the single bitstream as defined in the BoW PHY Specification

The interfaces between the various layers are defined as follows:

- System Interface - the interface to the SoC as defined by various bus protocols
- Transaction-Link Interface (TLI) - an intermediate interface between the transaction layer and the link layer (this interface is *not* defined by this specification and is *implementation- specific*)
- Link-Physical Interface (LPI) - the interface between the link layer and the physical layer based on the slice logic interface as defined in the BoW PHY Specification
- BoW Interface - the electrical interface between the TX and RX PHYs as defined in the BoW PHY Specification

The above layers and interfaces are illustrated below (layers in **bold** are described in this document):



ODSA D2D Layered Architecture

At present, the transaction layer and link layer are architected and optimized for the BoW standard. In the future, these layers may be adapted to other physical layers.

Instructions for Readers

Sections with *FIXME* in the title are in the process of being updated. Please consider the information in these sections to be subject to change.

References

- Bunch of Wires PHY Specification
  - https://opencomputeproject.github.io/ODSA-BoW/bow_specification.html

Overview

Terminology

This specification uses the following terminology (***FIXME:*** some definitions still to be added):

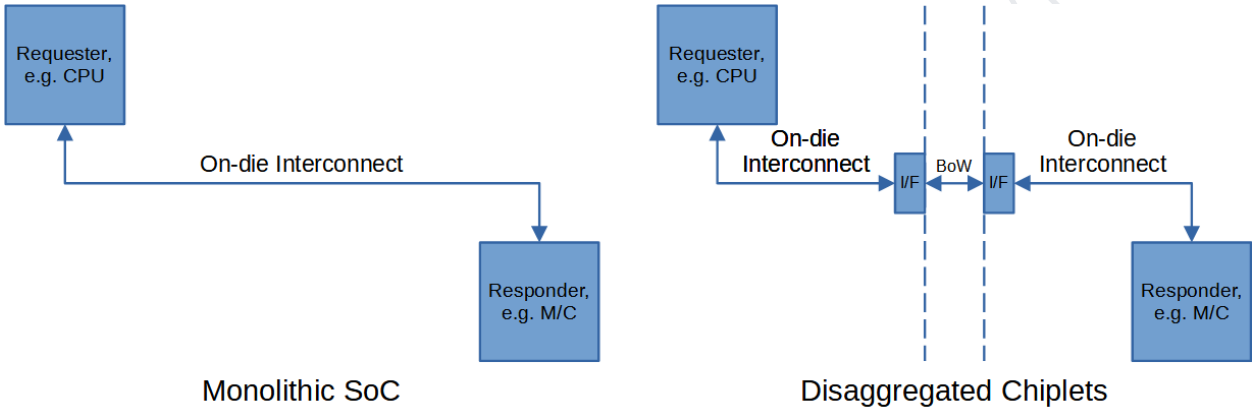
- *BoW interface* -
- *BoW slice* - the basic unit of a BoW PHY consisting of 16 data lanes and a forwarded source-synchronous differential clock

- *Bundle* - the unit of information transferred in a given direction (TX or RX) in a single cycle between the link layer and all BoW slices in the physical layer; a bundle consists of one or more fragments
- *Bus protocol channel* -
- *Bus protocol variant* -
- *D2D interface* -
- *Data rate* - the number of bits transferred per second per lane
- *Far-side* - for a pair of chiplets, the D2D interface on the other chiplet
- *FEC* - forward error correction
- *Fragment* - the unit of information transferred in a given direction in a single cycle between the link layer and a single BoW slice in the physical layer
- *Granule* - the 32b unit of quantization in the link layer
- *Interface profile* -
- *Lane* - a single data wire on a BoW slice
- *Link* - the logical connection between the link layers on two chiplets consisting of a number of near-side TX slices and an equal number of far-side RX slices plus the same or a different number of far-side TX slices and an equal number of near-side RX slices
- *Link layer* -
- *Link layer packet (LLP)* - the unit of information exchanged between the TX interface link layer and the RX interface link layer; an LLP consists of a 32b header and a 480b payload for a total of 16 granules
- *Link-physical interface (LPI)* -
- *Near-side* - for a pair of chiplets, the D2D interface on the given chiplet
- *On-die interconnect* -
- *Physical layer* -
- *Protocol layer* -
- *Protocol packet* - for a packetized protocol, the unit of information transferred in a given direction between the on-die interconnect and the D2D interface on a given protocol channel
- *Protocol transfer* - for a non-packetized protocol, the unit of information transferred in a given direction between the on-die interconnect and the D2D interface on a given protocol channel
- *Receive (RX) interface* - the portion of the D2D interface that receives BoW data from the far-side chiplet and transmits packets or transfers to an on-die interconnect
- *Receive (RX) slice* - a BoW slice on the RX interface
- *System interface* -
- *Transaction layer* -
- *Transaction layer packet (TLP)* - the unit of information exchanged between the TX interface transaction layer and the RX interface transaction layer; a TLP consists of a 12b header and a variable length payload; TLP transmission is governed by credit-based flow control
- *TLP Class* -
- *TLP Stream* -

- *TLP Type* -
- *Transaction-link interface (TLI)* -
- *Transmit (TX) interface* - the portion of the D2D interface that receives packets or transfers from an on-die interconnect and transmits BoW data to the far-side chiplet
- *Transmit (TX) slice* - a BoW slice on the TX interface

Background

The transaction and link layers, along with the BoW physical layer, create a complete D2D solution for die-disaggregation, where a monolithic SoC is divided into a set of constituent chiplets, as illustrated below:

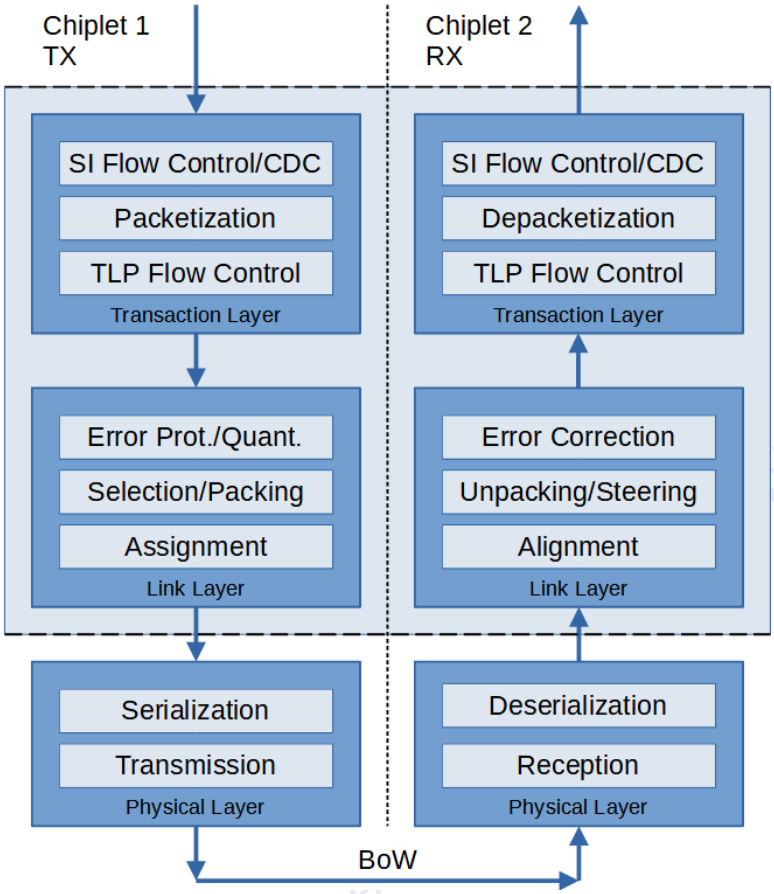


In order to ease system integration, the following goals were adopted for the D2D interface:

- Simplicity - build on the BoW physical layer, focus on die disaggregation (vs. package integration), and eliminate complexities of CDR, CRC/retry, etc.
- Low latency - enable aggressive implementation techniques, and use FEC to eliminate serialization overhead of CRC
- Scalability - support for different lane data rates and numbers of slices
- Portability - enable interfaces to be built with different implementation methodologies and in different process technologies
- Extensibility - create a modular framework that can easily add features over time

A D2D interface consists of a transmit interface (or *TX interface*) and a receive interface (or *RX interface*). On a given chiplet, the near-side TX interface transmits data to the far-side RX interface on the opposite chiplet, while the near-side RX interface receives data from the far-side TX interface on the opposite chiplet.

The following block diagram illustrates the TX interface on chiplet 1 and the RX interface on chiplet 2. More details follow.



D2D Interface Block Diagram

A D2D interface is conceptually a layered architecture. This specification defines the transaction and link layers, shown in the large light blue box above. The system interface, represented by the upper horizontal dashed line, is the boundary between the on-die interconnect and the transaction layer, while the link-physical interface (or LPI), represented by the lower horizontal dashed line, is the boundary between the link layer and the physical layer.

The following functions are performed by the transaction layer:

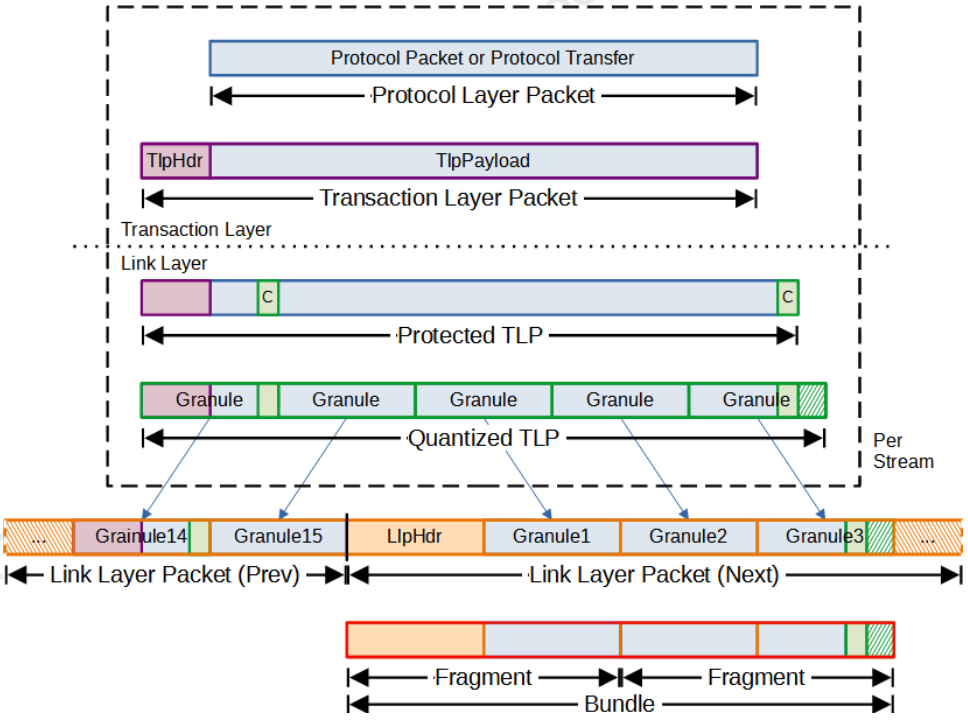
- TX Interface Transaction Layer
  - Implement bus protocol responder functionality and flow control on the system interface
  - Implement clock-domain crossings between the system interface clock domain and the TX interface clock domain
  - Packetize information received on the system interface into transaction layer packets (TLPs) and classify the TLPs into classes and streams
  - Manage flow control for each stream with the far-side RX interface
- RX Interface Transaction Layer
  - Manage flow control for each stream with the far-side TX interface
  - Depacketize TLPs into system interface information

- Implement clock-domain crossings between the RX interface clock domain and the system interface clock domain
- Implement bus protocol requester functionality and flow control on the system interface

In addition, the following functions are performed by the link layer:

- TX Interface Link Layer
  - Add error-correcting code (SECDED) and quantize TLPs
  - Select and pack eligible TLPs from different TLP streams into link layer packets (LLPs)
  - Assign LLP fragments to PHY slice logic interfaces
  - Drive the link layer training pattern
- RX Interface Link Layer
  - Correct single-bit errors and detect double-bit errors in TLPs
  - Align LLP fragments received from different PHY slice logic interfaces
  - Unpack TLPs from LLPs
  - Sample the link layer training pattern

Based on the above functions, the encapsulation steps in the transaction and link layers for various packets are illustrated in the following diagram:



Packet Formats

The units of information transferred on the system interface is known as a *protocol layer packet*. A protocol layer packet is encapsulated in a *transaction layer packet* (or *TLP*) that consists of a 12b header and a payload. Based on the interface profile, each TLP belongs to a TLP stream,

which is managed by credit-based flow control, and a TLP class, which represents a collection of TLP streams. Each TLP is protected with ECC check bits and is divided into 32b granules.

TLP granules from one or more TLP streams are packed efficiently into a 512b *link layer packet* (or LLP) that consists of a 32b header and 15 32b granules. An LLP is divided into fragments based on the number and serialization ratio of the BoW slices in the physical layer. The set of fragments transferred on the link-physical layer interface is known as a *bundle*.

The physical layer, which incorporates one or more BoW slices, forms the foundation for the transaction and link layers. Based on programmed ratios, each BoW TX slice serializes and each BoW RX slice deserializes its respective fragment relative to the BoW interface. While the link layer is responsible for aligning fragments received on different BoW slices, the physical layer is responsible for aligning individual lanes within a slice. Additional information may be found in the BoW PHY specification.

With the exception of any logic that may be implemented in the system interface clock domain, the entire D2D interface is rate matched. Consequently, flow control is only implemented in the transaction layer to avoid bus protocol deadlock due to head-of-line blocking (and to avoid receive buffer overruns in the transaction layer). If necessary, an asynchronous clock domain crossing between the on-die interconnect and the D2D interface may be implemented in the transaction layer.

The remainder of this specification provides additional details on the transaction layer and link layer.

Open Issues

- Complete definitions in the Terminology section
- Discovery for CRD (credit TLP) and MSG (message TLP) support
- Explicit scope of specification
- Add error model assumptions and uncorrected error probability analysis
- Add state transition diagrams in the reset section
- How much software standardization, e.g. memory-mapped register formats, addresses, etc.

Transaction Layer

The transaction layer is responsible for communicating information transferred on the system interface via *transaction layer packets*, or *TLPs*, on the transaction-link interface. A TLP is identified by its *TLP type* and consists of a TLP header, which contains type and auxiliary information, and a TLP payload, which contains bus protocol information. The minimum TLP payload size is 14b. One or more TLP types may be transferred within a *TLP stream*, which preserves the order of TLPs from the TX interface to the RX interface. Individual TLP streams implement credit-based flow control between the TX and RX interfaces and are non-blocking

with respect to each other. Finally, the set of TLP streams associated with a bus protocol is a *TLP class*. (See the Transaction Layer Packets section for more information.)

The transaction layer also incorporates a *virtual wire unit*, or *VWU*. In a TX interface, the VWU translates level-sensitive input wires on the system interface into TLPs, while in the RX interface, the VWU translates TLPs into level-sensitive output wires on the system interface. (See the Virtual Wire Packets section for more information.)

System Interface

The system interface transfers information between the transaction layer and the on-die system interconnect via one or more *bus protocols*. Bus protocols may define one or more *bus protocol channels* and some number of additional sideband signals, and each bus protocol may specify its own flow control mechanism, e.g. valid-ready signals, credit-based flow control, etc.

To support a particular bus protocol, the transaction layer must implement the required channels, sideband signals, and flow control mechanism for that bus protocol. In addition, the transaction layer must preserve any required ordering semantics across bus protocol channels and must guarantee forward progress of individual bus protocol channels where required.

Bus Protocol Variants

Given the wide range of on-die interconnect use-cases, bus protocols support a wide degree of configurability. For example, address and data widths may be different depending on the system requirements. To support this configurability, this specification introduces *bus protocol variants*, which specify the parameters of a bus protocol implementation.

Specifically, a bus protocol variant defines the following (see the section on Transaction Layer Packets for additional information about some of the terms below):

- The inclusion or exclusion of optional fields on the bus protocol channels
- The width of each field on the bus protocol channels
- The bundling and mapping of fields into the payloads of various TLP types
- The use of the auxiliary field in the TLP header for each TLP type
- The definition of any additional TLP types for control information
- The assignment of TLP types into TLP streams

Additionally, the set of TLP streams specified by a bus protocol variant effectively defines the corresponding TLP class.

When bus protocol channels are mapped to a TLP payload, the order of the signals that appear on the bus protocol channel does not have to match the order of bits in the TLP. A bus protocol variant defines the mapping between the system interface signals and the TLP payloads to ensure interoperability.

Interface Profiles

To ensure full interoperability between chiplets, this specification also introduces *interface profiles*. An interface profile defines the complete system interface, and D2D interfaces on different chiplets may interoperate if they support a common interface profile.

Specifically, an interface profile specifies the following (see the section on Transaction Layer Packets for more information):

- The inclusion of any standard bus protocol variants
- The definition of any custom bus protocol variants
- The combination of TLP streams from the supported bus protocol variants
- The support for credit (CRD) and message (MSG) TLPs
- The number and assignment of virtual wires and the reset state of each virtual wire

An interface profile may include or define one or more bus protocol variants. If more than one bus protocol variant is considered, the interface profile defines how the protocol channels from each bus protocol variant are combined. In particular, the interface profile may map the channels to separate TLP streams, or the interface profile may define modes so that the channels from different variants may share the same TLP stream. For example, an interface profile may support boot-time configuration of bus protocol A or bus protocol B, and in such a profile, the configuration would determine whether a given TLP type and stream corresponds to information from bus protocol A or bus protocol B.

Note: Protocol channels may share a TLP stream as long as lack of forward progress on one channel in the stream does *not* create a dependency that causes a deadlock in the system.

A D2D interface implementation may support one or more interface profiles, and an implementation is allowed to define custom interface profiles that may or may not conflict with the interface profiles standardized by the ODSA.

Note: This specification expects a rich variety of interface profiles to meet the requirements and needs of various use-cases. Interface profile specifications may be open or proprietary, and they may be more general or more specific. Over time, multiple ecosystems, based on the framework introduced in this document, are expected to arise.

Transaction Layer Packets

The TLP header and the various control and virtual wire TLPs are described below.

Header

A TLP header consists of 12b that identify the type of TLP and provide auxiliary control information. The format is illustrated below:

11 6	5	4 0
Type	*Rsv*	Aux

Note: The TlpHdr is likely to change from revision to revision. As a result, this specification is erring on the smaller side for Rev A.

The Type field encodes the TLP type, which implies the TLP stream and TLP class. For the common TLPs defined by this specification, the following table lists each TLP stream, the TLP type, and the TlpHdr.Type field encoding.

TLP Class	TLP Stream	TLP Type	TlpHdr.Type Encoding
Control	*None*	IDLE	`0x00`
		CRD	`0x01`
		MSG	`0x02`
Reserved	*Reserved*	*Reserved*	`0x03`
Virtual Wire	*None*	VWX	`0x04`
Reserved	*Reserved*	*Reserved*	`0x05`
		Reserved	`0x06`
		Reserved	`0x07`

Note: This specification will recommend standard TLP type encodings for common bus protocol channels; however, these may be modified by specifications for individual bus protocol variants and interface profiles.

Interface profiles may define additional TLP classes, streams, and types but *must not* use TLP type encodings `0x00` to `0x07`, which are reserved for this specification, and `0x38` to `0x3F`, which are reserved for custom interface profiles.

In general, TLP streams are managed by credit-based flow control, which prevents receive buffer overruns and ensures non-blocking communication. The control and virtual wire packets are guaranteed to be accepted by the RX interface, so those TLPs do not belong to a TLP stream and do not require flow control credits. (See the Flow Control section for more information.)

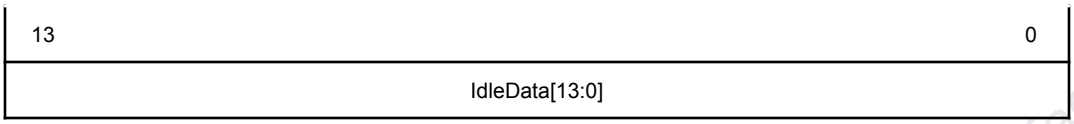
Packet Formats

This section describes the Aux field and TLP payload field for the defined TLPs.

Idle Packets

An idle TLP (IDLE) is inserted by the link layer to pad the remaining granules in an LLP (see here for more information). For IDLE TLPs, the Aux field is *reserved* and *must* be zeros.

The format of the TLP payload is illustrated below:

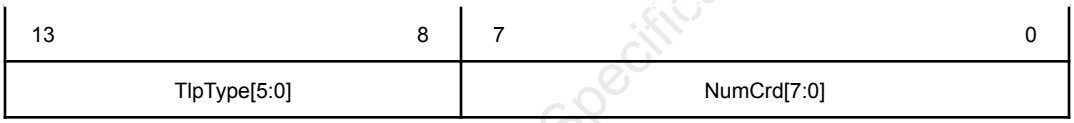


The 14b IdleData field is filled with all zeros.

Credit Packets

A credit TLP (CRD) grants up to 255 credits for a specific TLP type and support for CRD TLPs is *optional* for interface profiles that define alternate methods for exchanging credits, i.e. in the Aux field or in other TLPs. For CRD TLPs, the Aux field is *reserved* and *must* be zeros.

The format of the TLP payload is illustrated below:



The 6b TlpType field indicates the TLP type, which implies the TLP stream, while the 8b NumCrd field conveys the number of credits from 0 to 255 that have been granted for that stream. A CRD TLP that grants zero credits, i.e. NumCrd[7:0] = 0h00, is equivalent to an IDLE TLP.

An interface profile may also define credit TLPs for a TLP class. The format of those credit TLPs is defined by the interface profile.

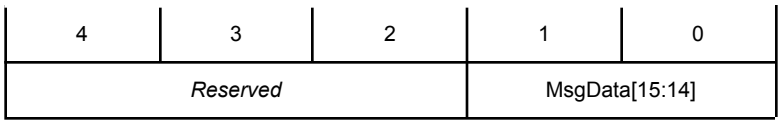
Note: See the Flow Control section for more information.

FIXME: Discovery for CRD support.

Message Packets

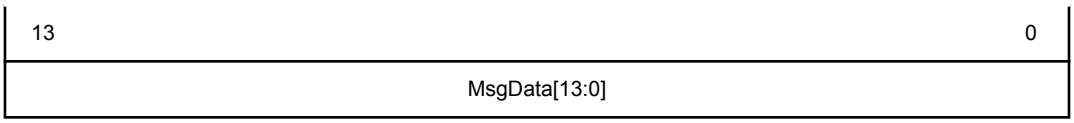
The message TLP (MSG) transmits miscellaneous information from the TX interface to the RX interface and support for MSG TLPs is *optional*. A MSG TLP is transmitted when 16b message data are written to a memory-mapped register in the TX interface. When a MSG TLP is received, the message data update a memory mapped-register in the RX interface.

For MSG TLPs, the Aux field contains 2b of message data:



The remaining bits in the Aux field are *reserved* and *must* be zero.

Additional message data is captured in the TLP payload and is illustrated below:



Note: The reserved Aux bits can be used to support up to 8 message types. Software or hardware may use these to exchange information in-band. For example, two message types could be used to implement a request-response protocol.

If a link layer implementation does *not* support MSG TLPs, the RX interface *must* ignore any received MSG TLPs.

FIXME: Discovery for MSG support.

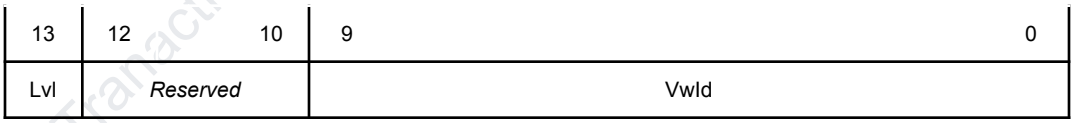
Virtual Wire Packets

The D2D interface supports the transmission of level-sensitive signal transitions on *virtual wires* from the TX interface to the RX interface. A maximum of 1024 virtual wires can be supported.

Input signals are connected to the TX interface, and output signals are connected to the RX interface. A high-to-low transition on an input signal causes a virtual wire low (VWL) TLP to be transmitted, while a low-to-high transition on an input signal causes a virtual wire high (VWH) TLP to be transmitted. When received, a VWL or VWH TLP causes the corresponding output wire on the RX interface to be driven to the level signaled by the TLP.

Collectively, VWL and VWH TLPs are classified as VWX TLPs, which are always transmitted and received in order for a given virtual wire. For VWX TLPs, the Aux field is *reserved* and *must* be zeros.

The format of the VWX TLP payload is illustrated below:



The Lvl bit in the VWX packet indicates the level being transmitted and distinguishes between VWL (Lvl = 0) and VWH (Lvl = 1) packets, while the VwId field identifies the virtual wire to which the level corresponds. The remaining bits are *reserved* and *must* be zeros.

For each virtual wire, the TX interface registers the most recent signal transition and transmits a virtual wire TLP corresponding to the transition. A signal transition is registered from the point the input signal changes state to the point the VWX TLP has been accepted by the link layer. If the input signal transitions while a previous transition is registered, a new virtual wire TLP,

corresponding to the new transition, is substituted for the registered virtual wire TLP. Once a virtual wire TLP is accepted by the link layer, the transition is no longer registered.

Note: The TX interface schedules TLPs for transmission based on changes in the state of the input signal, while the RX interface changes the state of the output wire based on the received TLP. As a result, a received TLP may *not* result in a state change of the output wire if the TLP matches the current state of the output signal.

The timing relationship among different input signal transitions on the TX interface is *not* maintained among the corresponding output signal transitions on the RX interface.

Additionally, a TX interface memory-mapped register provides the status of each input signal and the currently registered transition, while a RX interface memory-mapped register provides the status of each output signal.

Flow Control

To transmit a TLP type that belongs to a TLP stream, a TX interface requires a credit from the far-side RX interface on the link. Credits are granted by the far-side RX interface to the TX interface when the far-side RX interface is ready to accept a credited TLP. An RX interface may grant up to 255 credits per TLP stream and may grant different numbers of credits for different streams.

After link layer training (see Training) is complete, a near-side TX interface waits for credits from the far-side RX interface, which grants credits via the far-side TX interface and the near-side RX interface. Depending on the interface profile definition, credits are granted via credit packets, i.e. a CRD TLP or a credit TLP specified by the interface profile, or via the TlpHdr.Aux field. Once the credits are received, the near-side TX interface may begin transmitting TLP types associated with the credits.

For best performance, an implementation should transmit credits using the following rules:

- When transmitting a TLP type in a given TLP class, the TlpHdr.Aux field should signal as many credits as possible
- If credits are available to be transmitted, a credit TLP containing as many credits as possible should be transmitted in lieu of an IDLE TLP
- The TX interface should employ a mechanism to transmit credits from a given TLP class periodically if credits for that class cannot otherwise be transmitted

Link Layer

The link layer of the D2D interface is responsible for translating TLPs into *link layer packets*, or *LLPs*, and for mapping the LLPs onto the available physical layer PHY slices. This layer also performs error detection and correction.

The following sections describe individual functions as somewhat independent, but in practice, the functions are performed in parallel to achieve low latency.

TX Interface

Error Protection and Quantization

Each TLP, which consists of a TLP header and a TLP payload containing the protocol packet, is protected by SECDED ECC. The 12b TLP header and the first 14b of the TLP payload are encoded into a 32b *small codeword* containing 6 check bits. After the first 14 bits of the TLP payload are removed, any remaining TLP payload bits are divided into zero or more full groups of 120 bits and zero or one partial group of up to 119 bits. Each full group is encoded into a 128b *large codeword* containing 8 check bits.

The small codeword and the large codewords for full groups are calculated with the generation matrices below. To determine the check bits for a partial group, the TX interface appends zeros to pad the group to 120 bits and calculates the check bits based on the padded partial group. These zeros are not transmitted with the TLP, and the bits in the partial group and the calculated check bits for the padded partial group are concatenated and appended to the lower end of the TLP.

Note: As an error protection scheme, FEC/ECC has the following advantages:

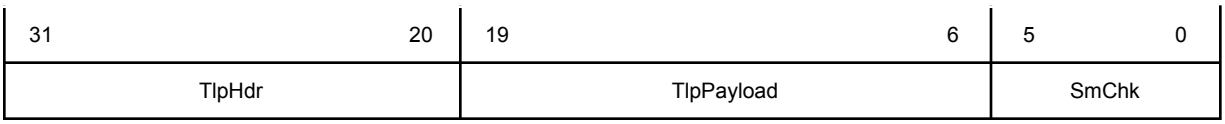
- Error detection and correction are performed in the RX interface, avoiding packet buffers in the TX interface and complex retry mechanisms for error correction
- Codewords can be computed simultaneously with selecting TLPs in the TX interface and independently from packing LLPs
- The RX interface can perform error detection on quantities smaller than an LLP and in parallel across TLPs

Note: A 128b codeword is the smallest codeword that does not affect the effective data bandwidth of the link given the decision to quantize TLPs into 32b granules (see below). Larger codewords reduce the number of check bits transmitted but do not increase the effective data bandwidth.

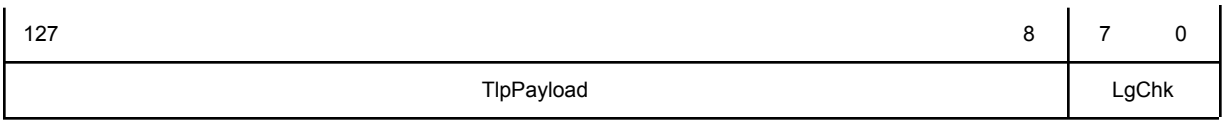
Note: With this scheme, the probability of an uncorrected error in one billion hours is less than 0.0005, assuming a 2 Tbps link with a raw BER of 10^{-15}.

FIXME: Add table of error probabilities and description of error model

The formats of the small and large codewords appear below:



Small Codeword (with TlpHdr)



Large Codeword

Note: In the case of a partial group, the TlpPayload for a large codeword may contain fewer than 120b; the omitted bits are assumed to be zero.

Once ECC has been added to a TLP, the protected TLP is quantized into 32b *granules*. If the protected TLP is not a multiple of 32b, the TX interface appends zeros to the protected TLP to extend the data to a multiple of 32b. Unlike the padding for ECC calculation, these zeros are transmitted with the TLP.

Note: This padding is not protected by ECC, and the bits are discarded by the RX interface.

The following table lists the relationship between the number of granules in a given TLP and the maximum number of payload bits in the TLP given the overheads introduced by adding the TLP header and ECC check bits.

TLP Granules	Max Payload (b)	TLP Granules	Max Payload (b)
1	14	17	494
2	38	18	518
3	70	19	550
4	102	20	582
5	134	21	614
6	158	22	638
7	190	23	670
8	222	24	702
9	254	25	734
10	278	26	758
11	310	27	790
12	342	28	822
13	374	29	854
14	398	30	878
15	430	31	910
16	462	32	942

Generation and Check Matrices

The small and large codewords are based on Hsiao codes, using a check matrix, C, and a generator matrix, G, which is a submatrix of C. The transmitted codeword is the concatenation of a dataword and additional check bits, calculated by multiplying the dataword by G. To determine any errors in a received codeword, a syndrome is calculated by multiplying the codeword by C. If the syndrome equals zero, no errors have occurred, but if the syndrome does not equal zero, the type of error depends on the parity of the syndrome. A syndrome with odd parity implies a single-bit, correctable error, and the value of the syndrome determines the bit in error. On the other hand, a syndrome with even parity implies a multi-bit, uncorrectable error.

The check matrix, C, for the small codeword appears in the table below. The generator matrix, G, corresponds to bits (columns) [31:6]. The final row shows the decimal syndrome values that correspond to an error in a given bit.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0
1	1	1	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0
1	1	0	1	1	1	0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1	0	1	0	0	0	1	0	0
1	0	1	1	1	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	1	0
0	1	1	1	1	1	0	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1
62	61	59	55	47	31	56	52	50	49	44	42	41	38	37	35	28	26	25	22	21	19	14	13	11	7	32	16	8	4	2	1

Small Codeword Bits [31:0]

The check matrix, C, for the large codeword appears in the four tables below. The generator matrix, G, corresponds to bits (columns) [127:8]. The final row in each table shows the decimal syndrome values that correspond to an error in a given bit.

127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1	1	1	0
1	1	0	1	1	1	1	1	0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1	0	1	1	0	0	1
1	0	1	1	1	1	1	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	1	0	1	0	1
0	1	1	1	1	1	1	1	0	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0
254	253	251	247	239	223	191	127	248	244	242	241	236	234	233	230	229	227	220	218	217	214	213	211	206	205	203	199	188	186	185	182

Large Codeword Bits [127:96]

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0
1	1	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	1	1	1	1	0	1
0	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1	0	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1
0	1	1	0	1	1	1	0	1	1	1	0	1	0	1	0	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1	0	0	1	0	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1	1	1
181	179	174	173	171	167	158	157	155	151	143	124	122	121	118	117	115	110	109	107	103	94	93	91	87	79	62	61	59	55	47	31

Large Codeword Bits [95:64]

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0
0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	0	0	1	0
0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1	0	0	0	1	0	0	0	1	0	0	1
0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1	0	0	0	0	1	0	0	0	1	0	0
224	208	200	196	194	193	176	168	164	162	161	152	148	146	145	140	138	137	134	133	131	112	104	100	98	97	88	84	82	81	76	74

Large Codeword Bits [63:32]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0
0	1	1	0	0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1	0	1	0	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	0	0	1	0
1	0	1	1	0	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	1
73	70	69	67	56	52	50	49	44	42	41	38	37	35	28	26	25	22	21	19	14	13	11	7	128	64	32	16	8	4	2	1

Large Codeword Bits [31:0]

Selection and Packing

A quantized TLP becomes eligible for transmission when a credit is available. The TX interface selects eligible, quantized TLPs to pack into an LLP, which consists of a 32b header and 480b of payload, for a total of 512b. The payload is packed as a series of TLP granules, subject to the following rules:

- LLP contents:
  - An LLP must contain a maximum of one TLP header per TLP stream
    - Stated differently, at least one LLP header must be appear between TLP headers from the same stream
  - An LLP must contain a maximum of one credit TLP per TLP class
  - An LLP must contain a maximum of one virtual wire TLP
  - An LLP may contain any number of IDLE TLPs
- LLP format:
  - A TLP may start on any LLP granule
  - A TLP must be transmitted in contiguous granules, including TLPs that span two (or more) LLPs
  - IDLE TLPs may be inserted between complete TLPs

Note: The above constraints imply that, when a TLP from a given stream spans LLPs, a new TLP from that stream can appear in the the same LLP as the remainder of the previous LLP.

To ensure forward progress of all streams, the selection of quantized TLPs *must* be round-robin.

An LLP has the following format:

511														0
HDR	G01	G02	G03	G04	G05	G06	G07	G08	G09	G10	G11	G12	G13	G14	G15

“Gxx” represents an LLP granule within the LLP. Higher order granules in a TLP are packed into higher order granules in an LLP, i.e. no granule swizzling occurs when packing an LLP. “HDR” represents the 32b LlpHdr field, which communicates the *start-of-packet* (*SOP*) information and has the following format:

31 22	21	20 6	5 0
Reserved	*Rsv*	TlpStart	HdrChk

Within the LlpHdr, the 15b TlpStart field indicates where a non-IDLE packet TlpHdr is located in the LLP, i.e. bit [20] corresponds to G01, bit [19] corresponds to G02, etc. The 6b HdrChk field

contains the check bits for the LlpHdr, i.e. the LlpHdr is effectively a small codeword (see Error Protection and Quantization above). All other bits are *reserved*.

Note: Bit [21] is *reserved* for a "continuation" bit from the previous LLP if that information proves to be valuable to an implementation.

Note: For IDLE packets, the corresponding TlpStart bit is *not* set in the LlpHdr. The RX interface determines which granules contain IDLE packets based on the state of the TlpStart bits and the number of TLP granules implied by the TlpHdr fields in those granules.

LLPs are presented to the physical layer in ascending granule order, i.e. HDR (or G00) is presented before G01, and G01 is presented before G12, etc. This order corresponds to the transfer order of the LLP.

Assignment

Once an LLP is packed, the granules in the LLP are assigned to *fragments*, which represent the data width of the interface on an individual PHY slice based on the PHY serialization ratio. The LLP header is always assigned to the least-significant bits, i.e. bits [31:0], of fragment 0, and the remainder of the LLP granules are assigned as described in LLP Transfer Order.

The number of active PHY slices determines the size of a fragment *bundle*, which represents the total number of bits transferred between the link layer and the physical layer per cycle on the link-physical interface. The *bundle type* is a function of the number of PHY slices and the size of each fragment, as illustrated in the table below.

		Number of PHY Slices		
		1	**2**	**4**
Fragment Size (Ser. Ratio)	**64b (4x)**	1x64b (64b)	2x64b (128b)	4x64b (256b)
	128b (8x)	1x128b (128b)	2x128b (256b)	4x128b (512b)
	256b (16x)	1x256b (256b)	2x256b (512b)	*4x256b (1024b)*

Fragment Bundle Types

Note: The 4x256b bundle type is *not* supported by Rev A.

The far-side TX and RX interfaces are required to support the same number of fragments; however, the fragment widths may differ in each to accommodate different process technologies and implementation methodologies. The assignment of granules to fragments is such that the LlpHdr (i.e. HDR) is located in the same bits on the link-physical interface, independent of the width of the fragment 0.

Note: Since the data rates of the TX interface and RX interface must match, an interface with a wider fragment must necessarily run at a lower frequency than an interface with a narrower fragment.

Implementation Note: To support a variety of data rates and possible implementations, an implementation *must* support all of these fragment configurations. The bundle type is boot-time configurable.

The transmit order for LLPs is described below.

LLP Transfer Order

The LLP transfer order depends on the width of the fragment. For the narrowest fragment size (i.e. 64b), granules are transmitted and received in order; however, for the wider fragment sizes (i.e. 128b or 256b), granules are interleaved to accommodate different fragment configurations on either the TX interface or the RX interface. Subsequent LLPs are transmitted or received in a repeating fashion.

Note: The transmit and receive order is arranged to avoid buffering LLP granules across clock cycles in either the TX or RX interface. Reordering of the granules is performed on the wider interfaces since those operate at a lower frequency for a given data rate.

The following subsections illustrate the LLP transmit and receive order for different fragment configurations, classified by the number of fragments.

Note: For the transfer order of granules within a fragment, see the Link-Physical Interface section.

One Fragment

Note: The color coding below applies to this section only. In the following tables, T indicates time in units of cycles, while FragN indicates a fragment number.

For a bundle type of 1x64b, an LLP is transmitted or received as follows:

T	Frag0	
0	G01	HDR
1	G03	G02
2	G05	G04
3	G07	G06
4	G09	G08
5	G11	G10
6	G13	G12

7	G15	G14

Bundle Type 1x64b

For a bundle type of 1x128b, an LLP is transmitted or received as follows:

T	Frag0			
0	G03	G02	G01	HDR
1	G07	G06	G05	G04
2	G11	G10	G09	G08
3	G15	G14	G13	G12

Bundle Type 1x128b

For a bundle type of 1x256b, an LLP is transmitted or received as follows:

T	Frag0							
0	G07	G06	G05	G04	G03	G02	G01	HDR
1	G15	G14	G13	G12	G11	G10	G09	G08

Bundle Type 1x256b

Two Fragments

Note: The color coding below applies to this section only. In the following tables, T indicates time in units of cycles, while FragN indicates a fragment number.

For a bundle type of 2x64b, an LLP is transmitted or received as follows:

T	Frag1		Frag0	
0	G03	G02	G01	HDR
1	G07	G06	G05	G04
2	G11	G10	G09	G08
3	G15	G14	G13	G12

Bundle Type 2x64b

For a bundle type of 2x128b, an LLP is transmitted or received as follows:

T	Frag1				Frag0			
0	G07	G06	G03	G02	G05	G04	G01	HDR

1	G15	G14	G11	G10	G13	G12	G09	G08

Bundle Type 2x128b

For a bundle type of 2x256b, an LLP is transmitted or received as follows:

T	Frag1								Frag0							
0	G15	G14	G11	G10	G07	G06	G03	G02	G13	G12	G09	G08	G05	G04	G01	HDR

Bundle Type 2x256b

Four Fragments

Note: The color coding below applies to this section only. In the following tables, T indicates time in units of cycles, while FragN indicates a fragment number.

For a bundle type of 4x64b, an LLP is transmitted or received as follows:

T	Frag3		Frag2		Frag1		Frag0	
0	G07	G06	G05	G04	G03	G02	G01	HDR
1	G15	G14	G13	G12	G11	G10	G09	G08

Bundle Type 4x64b

For a bundle type of 4x128b, an LLP is transmitted or received as follows:

T	Frag3				Frag2				Frag1				Frag0			
0	G15	G14	G07	G06	G13	G12	G05	G04	G11	G10	G03	G02	G09	G08	G01	HDR

Bundle Type 4x128b

RX Interface

Alignment

Based on the training patterns (see Training), the RX interface aligns the received fragments using a deskew FIFO to form a bundle. The RX interface supports the bundle types defined here, and the receive order for LLPs is described above.

For each LLP, the RX interface link layer first decodes the LlpHdr to determine where the granules containing a TlpHdr are located in the LLP, based on the TlpStart bits. Any errors in the LlpHdr *must* be corrected in order to determine the proper location of the TlpHdr granules.

Unpacking and Steering

Once the TlpHdr granules have been located, the TLPs are unpacked from the LLP. The small codewords containing the TlpHdr are corrected, if necessary, and the TLP granules are steered to a given TLP processing pipeline depending on the TLP stream identified by the header. VWX TLPs are steered to the appropriate virtual wire processing logic, and credit TLPs are steered to the near-side TX interface to grant credits.

IDLE TLPs between the end of a non-IDLE TLP and the start of another non-IDLE TLP are stripped.

Error Correction

Within a TLP processing pipeline, the large codewords are gathered and are corrected, as necessary. Once all the granules are gathered, the TLP is presented to the transaction layer.

The behavior of the RX interface in the presence of corrected and uncorrected errors is described in the Error Handling section.

Training

The link layer trains the D2D link to ensure the proper delivery of data. Performed after reset, training entails transmitting a known data pattern from the TX interface and aligning the data pattern in the RX interface. Alignment occurs at two levels:

- Granule phase alignment within a fragment (intra-slice)
- Fragment skew alignment within a bundle (inter-slice)

Granules within a fragment are serialized in the TX interface PHY slices and are deserialized in the RX interface PHY slices. Phase alignment ensures that the serialization and deserialization process is synchronized between the TX and RX interfaces so that the transfer order of granules within a sequence of fragments is preserved.

In addition, fragments within a bundle are transmitted on different paths. Variations in these paths may cause the granules transmitted in a given fragment to be received at different times and be aligned to different clock edges of a common clock relative to the granules in other fragments within the same transmitted bundle. Skew alignment ensures that granules in the fragments for a given transmitted bundle are aligned to the same received bundles. Based on the serialization and deserialization ratios, a transmitted bundle may represent part of a received bundle, a single received bundle, or multiple received bundles.

To adjust for phase and skew mismatches, the TX interface drives a sequence of 0 (0x00) to 255 (0xFF) in each byte of each granule of each fragment, e.g. 0x00000000 is driven in the first granule of each fragment, 0x01010101 is driven in the second, etc. The received sequence is sampled on the RX interface to determine the granule phase and fragment skew, as described in the following subsections.

Note: A 256b fragment consists of 8 32b granules, so the training sequence cycles through a minimum of 32 fragments.

Note: For a given slice, the BoW PHY is responsible for training the individual lanes with respect to the source-synchronous clock. The link layer assumes this level of training is performed prior to training for granule phase alignment and fragment skew alignment.

Granule Phase Alignment

The phase of granules within a fragment is determined by dividing the training pattern value in a given granule by the number of granules in the fragment. If the remainder of that computation is equal to the granule number in the fragment, the phase is correct.

Note: In the following diagrams, only the sequence number of the byte, rather than the full granule, is indicated for brevity. For example, 0x03 below implies 0x03030303 is transmitted in the granule.

For a 64b fragment, the phase is correct if

- Bit [0] of the training pattern value in bits [31:0] of the fragment equals 0b0, and
- Bit [0] of the training pattern value in bits [63:32] of the fragment equals 0b1.

T	Granule 1 [63:32]	Granule 0 [31:0]
0	0x01	0x00
1	0x03	0x02
2	0x05	0x04
3	0x07	0x06

Phase Aligned 64b Fragment

For a 128b fragment, the phase is correct if

- Bits [1:0] of the training pattern value in bits [31:0] of the fragment equals 0b00, and
- Bits [1:0] of the training pattern value in bits [63:32] of the fragment equals 0b01, and
- Bits [1:0] of the training pattern value in bits [95:64] of the fragment equals 0b10, and
- Bits [1:0] of the training pattern value in bits [127:96] of the fragment equals 0b11.

T	Granule 3 [127:96]	Granule 2 [95:64]	Granule 1 [63:32]	Granule 0 [31:0]
0	0x03	0x02	0x01	0x00
1	0x07	0x06	0x05	0x04

Phase Aligned 128b Fragment

Phase errors are corrected via an *implementation-specific* mechanism in the physical layer. The different types of phase errors received on the RX interface appear below:

T	Granule 1 [63:32]	Granule 0 [31:0]
0	0x00	0xFF
1	0x02	0x01
2	0x04	0x03
3	0x06	0x05
4	0x08	0x07

Phase Misaligned 64b Fragment +180°

T	Granule 3 [127:96]	Granule 2 [95:64]	Granule 1 [63:32]	Granule 0 [31:0]
0	0x02	0x01	0x00	0xFF
1	0x06	0x05	0x04	0x03
2	0x0A	0x09	0x08	0x07

Phase Misaligned 128b Fragment +90°

T	Granule 3 [127:96]	Granule 2 [95:64]	Granule 1 [63:32]	Granule 0 [31:0]
0	0x01	0x00	0xFF	0xFE
1	0x05	0x04	0x03	0x02
2	0x09	0x08	0x07	0x06

Phase Misaligned 128b Fragment +180°

T	Granule 3 [127:96]	Granule 2 [95:64]	Granule 1 [63:32]	Granule 0 [31:0]
0	0x00	0xFF	0xFE	0xFD
1	0x04	0x03	0x02	0x01
2	0x08	0x07	0x06	0x05

Phase Misaligned 128b Fragment +270°

For a 256b fragment, the phase is correct if

- Bits [2:0] of the training pattern value in bits [31:0] of the fragment equals 0b000, and
- Bits [2:0] of the training pattern value in bits [63:32] of the fragment equals 0b001, and
- Bits [2:0] of the training pattern value in bits [95:64] of the fragment equals 0b010, and
- Bits [2:0] of the training pattern value in bits [127:96] of the fragment equals 0b011, and
- Bits [2:0] of the training pattern value in bits [159:128] of the fragment equals 0b100, and
- Bits [2:0] of the training pattern value in bits [191:160] of the fragment equals 0b101, and
- Bits [2:0] of the training pattern value in bits [223:192] of the fragment equals 0b110, and
- Bits [2:0] of the training pattern value in bits [255:224] of the fragment equals 0b111.

T	Granule 7 [255:224]	Granule 6 [223:192]	Granule 5 [191:160]	Granule 4 [159:128]	Granule 3 [127:96]	Granule 2 [95:64]	Granule 1 [63:32]	Granule 0 [31:0]
0	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00

Phase Aligned 256b Fragment

Phase errors are corrected via a TBD mechanism. The different types of phase errors received on the RX interface are similar to those described above for 64b and 128b fragments.

Fragment Skew Alignment

The relative skew between fragments is determined by comparing the training pattern values in granules of the two fragments. If the training pattern values are equal in the corresponding granules in each fragment, the fragments are aligned. This alignment is illustrated below:

T	Fragment N+1		Fragment N	
0	0x01	0x00	0x01	0x00
1	0x03	0x02	0x03	0x02
2	0x05	0x04	0x05	0x04
3	0x07	0x06	0x07	0x06

Skew Aligned 64b Fragments

T	Fragment N+1				Fragment N			
0	0x03	0x02	0x01	0x00	0x03	0x02	0x01	0x00
1	0x07	0x06	0x05	0x04	0x07	0x06	0x05	0x04

Skew Aligned 128b Fragments TX/RX Pattern

T	Fragment N+1				Fragment N			
0	0x07 0x06	0x05 0x04	0x03 0x02	0x01 0x00	0x07 0x06	0x05 0x04	0x03 0x02	0x01 0x00

Skew Aligned 256b Fragments TX/RX Pattern

Note: Each cell for the 256b fragment pattern represents a pair of granules.

Assuming fragments are synchronized to a common clock in the RX interface. examples of single cycle skew errors on the RX interface are illustrated below:

T	Fragment N+1		Fragment N	
0	0xFF	0xFE	0x01	0x00
1	0x01	0x00	0x03	0x02
2	0x03	0x02	0x05	0x04
3	0x05	0x04	0x07	0x06
4	0x07	0x06	0x09	0x08

Single-cycle Skew Misaligned 64b Fragments (N+1 arrives late)

T	Fragment N+1				Fragment N			
0	0x03	0x02	0x01	0x00	0xFF	0xFE	0xFD	0xFC
1	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00
2	0x0B	0x0A	0x09	0x08	0x07	0x06	0x05	0x04

Single-cycle Skew Misaligned 128b Fragments (N arrives late)

Skew errors are corrected by adjusting the delay a given fragment relative to other fragments (see the subsequent section for more details).

Deskew FIFO

Skew alignment is achieved via a *deskew FIFO* in the RX interface. The deskew FIFO synchronizes the received fragments to a common clock and allows the delay of individual fragments to be increased or decreased with respect to other fragments. Synchronization and

delay adjustments are performed by an *implementation-specific* mechanism. The width of the deskew FIFO matches the width of the maximum bundle type supported on the LPI.

When the RX interface is in the RX_IDLE state (see Reset), the deskew FIFO *must* ignore the fragments received from the PHY slices; however, when the RX interface is in the RX_TRAIN state (or any state other than RX_IDLE), the deskew FIFO captures the received fragments. In both the RX_IDLE and RX_TRAIN states, the RX interface ignores the output of the deskew FIFO, though in the RX_TRAIN state, the output is sampled periodically by training logic to determine the phase alignment and the skew alignment of the training pattern.

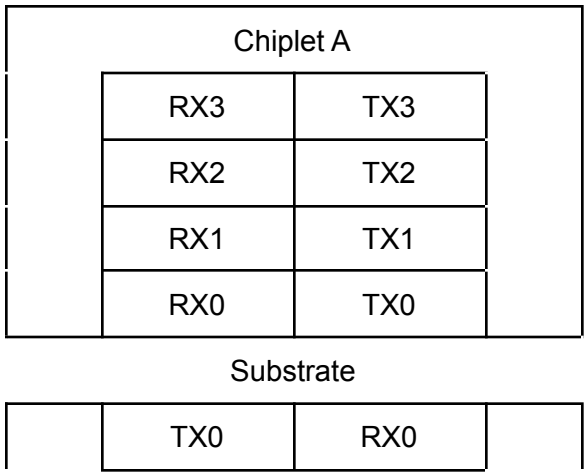
The relative skew between fragments M and N can be determined by comparing the training pattern values in byte 0 of each fragment. If the values are equal, the fragments are aligned. If the value in fragment M (e.g. 0x08) follows the value of fragment N (e.g. 0x04) in the training pattern, the delay of fragment M is increased or the delay of fragment N is decreased; on the other hand, if the value in fragment M (e.g. 0x00) precedes the value of fragment N (e.g. 0x04) in the training pattern, the delay of fragment M is decreased or the delay of fragment N is increased. Although the amount of delay is generally equal to the relative difference between the two values divided by the number of granules in the fragments, care must be taken when the values may have wrapped from 0xFF to 0x00.

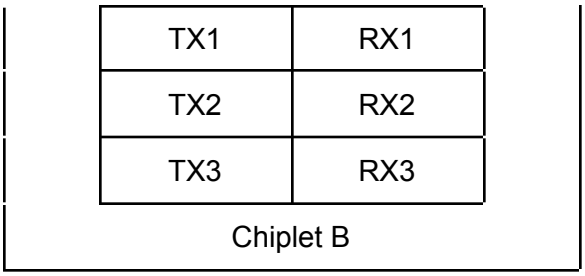
Implementations that support only 1x bundle types do not require a deskew FIFO.

Link-Physical Interface

The link-physical interface consists of one, two, or four slice logic interfaces, as defined by the BoW interface specification, and the link layer supports serialization and deserialization ratios of 4x, 8x, and 16x, which result in 64b, 128b, and 256b fragments, respectively. Fragments are transferred on the least significant bits of the slice logic interface, and unused bits on the LPI are driven to zero by the TX interface and ignored by the RX interface.

To enable interoperability, the link layer supports boot-time configuration of the number of active slices and assumes the following physical organization and labeling of the individual slices (assuming north-south orientation):





The following slices are active for a given number of active slices:

- One active slice: TX0 and RX0
- Two active slices: TX0/TX1 and RX0/RX1
- Four active slices: TX0/TX1/TX2/TX3 and RX0/RX1/RX2/RX3

As described in LLP Transfer Order, Frag0 corresponds to TX0 and RX0, Frag1 corresponds to TX1 and RX1, etc.

Finally, the following table indicates the unidirectional link bandwidth achievable with different data rates and serialization ratios:

Data Rate 1 lane	Ser/Des Ratio	Slice Logic I/F	LL Freq.	BW 1 Slice	BW 2 Slices	BW 4 Slices
2 Gbps	16x	[255:0]	125 MHz	32 Gbps	64 Gbps	*N/A*
	8x	[127:0]	250 MHz			128 Gbps
	4x	[63:0]	500 MHz			
4 Gbps	16x	[255:0]	250 MHz	64 Gbps	128 Gbps	*N/A*
	8x	[127:0]	500 MHz			256 Gbps
	4x	[63:0]	1 GHz			
8 Gbps	16x	[255:0]	500 MHz	128 Gbps	256 Gbps	*N/A*
	8x	[127:0]	1 GHz			512 Gbps
	4x	[63:0]	2 GHz			
16 Gbps	16x	[255:0]	1 GHz	256 Gbps	512 Gbps	*N/A*
	8x	[127:0]	2 GHz			1024 Gbps
	4x	[63:0]	*N/A*	*N/A*	*N/A*	*N/A*

Cells with unsupported combinations of data rates, serialization/deserialization ratios, etc. are shown in gray. An implementation must support all data rates but may support a subset of the serialization and deserialization ratios.

The LPI on the far-side RX interface must preserve the order of fragments and the order of bits within each fragment as presented on the LPI by the near-side TX interface.

Memory-Mapped Register State

TX Interface

The memory mapped register state in the TX interface is classified into groups by function below. Major bullets indicate the necessary bits and fields, while sub-bullets provide more information about the bits and fields.

Interface Control and Status

Note: In this section, state may be implemented in a single register or in multiple registers.

- Credit reset
  - See Link Reset
- PHY slice reset
  - Support for up to 4 slices (1b per slice)
- PHY slice ready
  - Support for up to 4 slices (1b per slice)
- Active slices
  - 1 slice (0b00)
  - 2 slices (0b01)
  - 4 slices (0b11)
- Fragment size
  - 64b (0b00)
  - 128b (0b01)
  - 256b (0b10)
- State
  - TX_IDLE (0b00) - transmitting idle LLPs
  - TX_TRAIN (0b01) - transmitting training pattern
  - TX_RUN (0b11) - transmitting data

Messages

Note: Link layer support for messages is *optional*; however, this specification envisions up to eight message registers.

- TX Message
  - A write to this register generates a MSG TLP
  - Message data (16b) - the TX interface inserts this field into the message data
  - Other control bits may be defined

Virtual Wire Inputs

Note: In this section, state is defined for each specified virtual wire. State corresponding to virtual wires that are *not* implemented is *Reserved*.

- Disable (32x 32b read/write registers)
    - If clear, a transition on the virtual wire input is registered
    - If set, a transition on the virtual wire input is *not* registered
        - A transition from 1-to-0 on the disable bit registers the current state of the virtual wire input and causes a corresponding VWX TLP to be transmitted
    - Reset to 1

RX Interface

The memory mapped register state in the RX interface is classified into groups by function below. Major bullets indicate the necessary bits and fields, while sub-bullets provide more information about the bits and fields.

Interface Control and Status

Note: In this section, state may be implemented in a single register or in multiple registers.

- Credit reset
    - See Link Reset
- PHY slice reset
    - Support for up to 4 slices (1b per slice)
- PHY slice ready
    - Support for up to 4 slices (1b per slice)
- Active slices
    - 1 slice (0b00)
    - 2 slices (0b01)
    - 4 slices (0b11)
- Fragment size
    - 64b (0b00)
    - 128b (0b01)
    - 256b (0b10)
- State
    - RX_IDLE (0b00) - ignoring data
    - RX_TRAIN (0b01) - sampling training pattern
    - RX_WAIT (0b10) - waiting for sync LLP
    - RX_RUN (0b11) - receiving data

Messages

Note: Link layer support for messages is *optional*; however, this specification envisions up to eight message registers.

- RX Message
    - A MSG TLP fills this register
    - Message data (16b) - the RX interface captures the message data into this field
    - Other control bits may be defined

Virtual Wire Outputs

Note: In this section, state is defined for each specified virtual wire. State corresponding to virtual wires that are *not* implemented is *Reserved*.

- Disable (32x 32b read/write registers)
    - If clear, receipt of a VWX TLP modifies the virtual wire output accordingly
    - If set, receipt of a VWX TLP does *not* modify the virtual wire output
    - Reset to 1

The reset state, 0 or 1, for each virtual wire output is defined by the interface profile and should match the reset state of the corresponding virtual wire input on the far-side TX interface.

Error Handling

The RX interface detects errors in LLP headers, TLP headers, and TLP payloads based on the received ECC codewords. The ECC algorithm supports detection and correction of single-bit errors and detection of all double-bit errors and certain additional multi-bit errors. Combining both factors, errors are classified as one of the following:

- Corrected error
    - LLP header (32b ECC codeword)
    - TLP header (32b ECC codeword)
    - TLP payload (128b ECC codeword)
- Uncorrected error
    - LLP header (32b ECC codeword)
    - TLP header (32b ECC codeword)
    - TLP payload (128b ECC codeword)

The signaling and handling of these errors in a system is *implementation-defined*.

Critical errors imply the following loss of payload and/or credits, depending on the location of the error:

- LLP header - loss of payload for multiple unknown streams and loss of credits for multiple unknown streams

- TLP header - loss of payload for a single unknown stream and loss of credits for multiple unknown streams
- TLP payload - loss of payload for a single known stream

The loss of any information on the D2D interface is likely to result in system failure, and a system reboot may be required to return to normal operation. To avoid propagating corrupted payload information, the RX interface drops the following granules based on the location of the error:

- LLP header:
    - All granules in the current LLP after any granules from a previous TLP that continues into this LLP
    - All granules in subsequent LLPs up to the next TLP header as indicated by the next TlpStart bit in a subsequent error-free LLP header
    - ***Note:*** Basically, the granules that are discarded start with the granule after the last known good TLP in a previous LLP and end with the granule before the next known good TLP in a subsequent LLP
- TLP header:
    - All granules from the TLP header with the error up to the next TLP header as indicated by the next TlpStart bit in the current LLP or a subsequent LLP
- TLP payload:
    - All granules from the current TLP; however, the information in the TlpHdr field is preserved
    - ***Note:*** In the case of TLP payload errors, the TLP header information is still valid, so the near-side credit corresponding to the stream and the far-side credits in the header are not lost

Granules between valid TLPs *must* also be checked for errors. Errors in these granules are considered to be corrected errors whether the detected errors are single-bit, double-bit, or more.

Note: In general, these granules are filled with IDLE TLPs, but errors may change the decode of the TLP. The RX interface must use the framing information provided by the LLP and TLP headers to determine whether the intervening granules are valid.

Error Injection Registers

FIXME

Error Logging Registers

FIXME

Reset

The D2D interface supports two types of reset states: hard reset and link reset. These reset states are defined by various combinations of external signals and internal state.

Hard Reset

An implementation may have one or more external reset signals that initialize different logic blocks. The *hard reset* state is characterized by the assertion of all implemented external reset signals. In the hard reset state, all state required for specified and deterministic operation of the transaction layer, link layer, and physical layer must be initialized. If any TX or RX interface enters the hard reset state, all TX and RX interfaces on the link *must* also enter the hard reset state eventually. The hard reset state also puts a TX or RX interface into the link reset state.

Once all TX and RX interfaces have been in the hard reset state for an *implementation-specific* amount of time, the external reset signals may be deasserted, after which point the TX and RX interfaces remain in the link reset state. Before link training is performed, the physical layer must be initialized and the LPI must be configured based on the number of active slices, the serialization or deserialization ratio, etc.

Link Reset

A TX interface is in the *link reset* state when all of the following conditions are true:

- The TX state is TX_IDLE
- The TX credit reset bit is set
- The TX virtual wire inputs are disabled

In the TX_IDLE state, the TX interface transmits *idle LLPs*, or LLPs consisting of only IDLE TLPs. In addition, when the TX credit reset bit is set in a TX interface, the credit counters are reset to zero, i.e. no credits have been granted to the TX interface.

The RX interface is in the link reset state when all of the following conditions are true:

- The RX state is RX_IDLE
- The RX credit reset bit is set
- The RX virtual wire outputs are disabled

In the RX_IDLE state, the RX interface *must not* sample the received fragments from the slices, i.e. registers must not capture data. In addition, when the credit reset bit is set in an RX interface, the credit counters are reset to an *implementation-specific* number, i.e. all credits have been returned to the RX interface.

Note: The TX and RX interface states are set independently of the TX and RX credit reset bits.

Note: The hard reset state initializes the state that causes the TX and RX interfaces to be in the link reset state.

If any TX or RX interface enters the link reset state, all TX and RX interfaces on the link *must* also enter the link reset state eventually.

While in the link reset state, the memory-mapped registers in the TX and RX interfaces *must* be accessible so link training may be performed and the link reset state can be exited.

Training and Link Reset Exit

FIXME: Add state transition diagram

RX_IDLE → RX_TRAIN and TX_IDLE → TX_TRAIN

To begin training a link, each RX interface transitions to the RX_TRAIN state, activating the deskew FIFO in each interface, and then each TX interface transitions to the TX_TRAIN state, initiating the transmission of the training pattern. Samples are taken from the output of the deskew FIFO, and the phase of the granules and the skew of the fragments are adjusted (see the Deskew FIFO section). This process repeats until both the phase and skew are aligned; however, if alignment cannot be achieved in either RX interface, the TX and RX interfaces on the link are put into the link reset state, and system-specific mitigations are taken.

TX_TRAIN → TX_IDLE and RX_TRAIN → RX_WAIT

Once alignment has been achieved, each TX interface transitions to the TX_IDLE state and clears each TX credit reset bit. After the idle LLPs have propagated to the far-side RX interfaces, each RX interface transitions to the RX_WAIT state and clears each RX credit reset bit. At this point, an RX interface begins transferring credits to its near-side TX interface.

Note: An RX interface *must not* transition to RX_WAIT until idle LLPs have been received. This occurs typically within 100 RX link layer clocks; however, the deskew FIFO output can be sampled in the RX_TRAIN state to ensure that idle LLPs are being received.

TX_IDLE → TX_RUN and RX_WAIT → RX_RUN

After each RX interface has transitioned to the RX_WAIT state, each TX interface transitions to the TX_RUN state, at which point the TX interfaces may begin transmitting LLPs with non-IDLE TLPs. The first LLP transmitted with a non-IDLE TLP is known as the *sync LLP*, and transmission of the sync LLP *must* occur in such a manner that the LLP header is received in granule 0 of fragment 0, independent of the RX interface bundle type.

Note: Typically, the first non-IDLE TLP will be a credit TLP; however, any TLP may be the first non-IDLE TLP.

While the RX interface is in the RX_WAIT state, an LLP header synchronization state machine in the RX interface checks each received bundle for the sync LLP. Once the sync LLP is detected, the state machine locks on the LLP header based on the RX interface bundle type,

and the RX interface transitions to the RX_RUN state. Otherwise, the RX interface remains in the RX_WAIT state and checks the next received bundle.

Once all TX and RX interfaces are in the TX_RUN and RX_RUN states, respectively, the virtual wire outputs on each RX interface are enabled, and then the virtual wire inputs on each TX interface are enabled.

At this point, the link is fully operational.

Link Reset Entry

FIXME: Add state transition diagram. Note there still remains some unspecified behavior regarding what state the virtual wires should be when the disable bits are set.

To put the TX and RX interfaces into the link reset state, the above process is roughly reversed.

First, for each TX interface, the virtual wire inputs are disabled, and the state transitions to TX_IDLE. On the TX_RUN to TX_IDLE transition, a TX interface *must* complete sending any LLPs or TLPs that are partially transmitted, and once those have been transmitted, the TX interface no longer accepts TLPs and begins transmitting idle LLPs. Once each RX interface begins receiving the idle LLPs, the virtual wire outputs are disabled, and the RX interface transitions to the RX_IDLE state.

Once the TX interfaces and the RX interfaces are in the TX_IDLE and RX_IDLE states, respectively, the link is idle, and no more credits are exchanged between the near-side TX interface and far-side RX interface (though credits may continue to propagate from the RX interface to the TX interface on the near-side). At this point, all TX and RX interfaces enter into the link reset state once all the corresponding credit reset bits are set.

Note: Setting the credit reset bit resets the state of credits, so there is no requirement to ensure that credits have been exchanged before the TX interface begins transmitting idle LLPs.

Clocking

Clocking in the transaction and link layers is a function of the data rate of the link, the serialization ratio on the TX interface, and the deserialization ratio of the RX interface. On the same side of the link, the TX and RX interfaces must have the same serialization and deserialization ratios, respectively; however, on the opposite sides of the link, the TX and RX interfaces may have different ratios.

On the TX interface LPI, the physical layer provides one *TX slice clock* per fragment at a frequency determined by the serialization ratio, and all TX slice clocks must be synchronous with respect to each other. On the RX interface LPI, the physical layer provides one *RX slice clock* per fragment at a frequency determined by the deserialization ratio, but in this case, the RX slice clocks are mesochronous with respect to each other. The RX interface implements an *RX common clock* to which all received fragments are synchronized (see the Deskew FIFO section for more details). The TX slice clocks in the TX interfaces on opposite sides of the link

must share a common point, i.e., at a minimum, there must be a shared reference clock source for the PLLs that generate the TX slice clocks for the TX interfaces. Consequently, within the TX and RX interfaces, all the slice clocks are either synchronous or mesochronous with respect to each other, and the RX common clock may be synchronous, mesochronous, or asynchronous with respect to the slice clocks.

Finally, the on-die interconnect may provide one or more *system clocks* on the system interface. The system clocks may be synchronous, mesochronous, or asynchronous with respect to the TX slice clocks and the RX common clock. If necessary, the transaction layer implements a clock domain crossing between the system clocks and the TX slice clocks and between the system clocks and the RX common clock.

The following table summarizes these relationships, where S indicates a synchronous relationship, M indicates a mesochronous relationship, and A indicates an asynchronous relationship:

Clock	TX slice clock	RX slice clock	RX common clock	System clock
TX slice clock	S	M	SMA	SMA
RX slice clock	M	M	SMA	*N/A*
RX common clock	SMA	SMA	S	SMA
System clock	SMA	*N/A*	SMA	SMA